DEGREE OF MASTER OF SCIENCE IN FINANCIAL ECONOMICS

[1cm]

FINANCIAL ECONOMETRICS

HILARY TERM 2021 COMPUTATIONAL ASSIGNMENT 3
FORMAL WORK 1

February 2021

Assignment must be submitted before
noon (12:00) March 19 2019 (9th Week)
by uploading to SAMS**.**

This is group work. Groups of 3 or 4 are permitted.
Groups with less than 3 or more than 4 are not permitted without explicit permission.
All solutions must be submitted by the due date and time.
Do not write the names of members of your group on your submission.

*Candidates should answer **all** questions.*
Suggested Length: 10 pages; limit 15 pages/3,750 words.
Material on pages 16+ will not be assessed.
Limit does not include the cover sheet, academic honesty declaration or submitted code files.
All material, including figures, equations, and explanatory text must fit within 15 pages.

turn over

Installing the arch package

The arch package is not part of Anaconda and so must be installed directly. The simplest way to install the package is

```
%pip install arch --upgrade
```

in a jupyter cell and run the cell to install it. You only need to do this once (you do not need to rerun after the initial install). An alternative way to install the arch package is to open an Anaconda command prompt and enter.

```
pip install arch --upgrade
```

A video demonstration of the alternative approaches available to install the arch package is available.

Assessment

This assignment is assessed in 2 parts:

- 67% - Report. This report should focus on analysis and synthesis and not code or the numerical values of the problems.

- 33% - Autograder. 3 functions must be submitted to compute the required outputs using the inputs. The signature of each function is provided as part of the problem. You must ***exactly*** match the function name. Submissions must be in Python and must be in a single Python file (*some_filename*.py) containing all functions. IPython notebooks are **not** accepted. Please pay close attention to the dimensions and types of each input or output. All data inputs will be pandas DataFrames or Series, as indicated in the program description.
  Note: *Please* run your code in the function you submit to ensure that it does not produce an error. A function that errors when run is given a mark of 0. The autograder uses loose criteria when judging correctness, and so any value within about 1% of the reference will be marked as correct. See the example file solutions-fw1.py for the structure of the file expected by the autograder.

Tips for Autograded Code

- Your submissions **MUST** be a Python file (.py). IPython notebooks are not accepted, and submitting your solution in a notebook will result in a mark of 0.

- Your submission will not have access to any data files. You must not read or write anything from your program.

- Your submission will be run in a random directory. You cannot assume anything about data files existing in any particular location. Your code does not need to access data. All required data is passed through the inputs.

- You can use demo-autograder-fw1.py to check that your solution accepts the required arguments and returns values with the correct type and shape. You should run the program along with your code *in an empty directory*, which is how the autograder runs it. Code similar to

  ```
  mkdir empty
  cd empty
  <copy demo-autograder-fw1.py to empty along with your submission>
  <edit demo-autograder-fw1.py to give it the name of your submission>
  python demo-autograder-fw1.py
  ```

  can be used to check that your code will run correctly.

- The autograder submission should not normally have code that you wrote as part of the analysis. It should only contain the required functions, imports, and code necessary to run the functions. In *ideal* submission would not execute any code when imported, and instead would only contain import statements and functions.

```
import numpy as np
import pandas as pd

def first_function(a, b):
    # Do something
    return "something"

def used_by_second_function(x, y):
    # Do something
    return "something", "else"

def used_by_second_function(x, y, z):
    w = used_by_second_function(x, y)
    return w[0], w[1], z
```

Problem 1

1. Download the available high-frequency **price data**. This file contains 10 years of 1-second **price data**. Import this into your analysis software. Each column contains the data from a single day. The row index is stored as seconds past midnight (integers). You can load the data using:

```
prices = pd.read_csv("mfe-formal-work-1-2020-2021.csv.gz",
                     index_col="time",
                     compression="infer")
prices.columns = pd.to_datetime(prices.columns)
```

   .

2. What is the optimal sampling time for realized variance?

3. What is the optimal sampling time for RV^{AC_1}?

4. Using the optimal RV sampling scheme you arrived at in 2, estimate a HAR with 1-, 5- and 22-lags using 50% of the sample.

5. Estimate a GARCH(1,1) or GJR-GARCH(1,1) (as appropriate) using the same 50% of the sample using end-of-day data.

6. Estimate the same model you produced in 5 using the pseudo returns constructed from the realized variance covered in the lecture slides (use 50% of the sample).

7. Compare the models in 4, 5, and 6 across 1, 5 and 22-day horizons. Use the $RV + r^2_{CtO}$ to evaluate your model. Note: The models in 4 and 6 only use open-to-close data, and so you should consider whether you need to adjust the forecasts to match the variance over the entire day, which is what the evaluation measure captures.

Problem 2

Forecasting persistent time series is hard. Design and implement a Monte Carlo experiment to determine for which values of $\hat{\rho}$ it is better to assume that the true value of $\rho = 1$ instead of using the estimated value when forecasting the time series. You should simulate data from the model

$$y_t = \rho y_{t-1} + \varepsilon_t \tag{1}$$

where $\varepsilon_t \overset{iid}{\sim} N(0,1)$ for values of $\rho \in \{0.90, 0.91, \ldots, 0.99\}$ and $y_0 \sim N(0, {}^{1}/_{1-\rho^2})$. You should allow the model to have a constant, so that you estimate

$$y_t = \phi_0 + \phi_1 y_{t-1} + \varepsilon_t.$$

Each time you simulate a time-series and estimate a model you get the estimated parameter values. You can then compare the forecast produced using the estimated parameters against a random walk forecast (which has $\phi_0 = 0$ and $\phi_1 = 1$) using out-of-sample values. Use sample sizes of 100, 250 and 500 observations where you use 50% of the simulated data to estimate parameters and the other half to evaluate the model. You should have at least 1000 simulated runs of each configuration.

1. For each of the sample sizes, describe the probability that the random walk forecast outperforms the estimated parameter forecast *as a function of* $\hat{\phi}_1$. Use your analysis to determine the value $\hat{\phi}_1$ at which it is better, on average, to ignore your parameter estimate and use the simple RW model. Your analysis should indicate how sample size affects your conclusion.

2. Suppose you pre-test using an ADF for a unit root. Does a standard test size of 5% to select between the RW and the stationary AR improve forecast performance?

3. Recall that AR models can be estimated using OLS. Describe how optimal shrinkage of ϕ_1 to 1 could be implemented. If you implement shrinkage here, what do you find for the optimal shrinkage as a function of the OLS estimate $\hat{\phi}_1$? Note: This question is intentionally challenging and it requires you to synthesize information from different parts of the course.

Notes

You should use `statsmodels.tsa.api.AutoReg` not `SARIMAX` when studying the problem. `AutoReg` uses OLS and is orders of magnitude faster than `SARIMAX` at fitting AR models.

Code Problems

Weighted Historical Simulation

Implement Weighted Historical Simulation p-VaR using a recursive generation scheme. The p-Value-at-Risk estimator is defined in a sample size of T a

$$\begin{aligned} p - VaR_T &= -\min_r \hat{G}_T(r) \text{ such that } \hat{G}_T(r) \geq p \\ \hat{G}_T(r) &= \sum_{t=T-\tau+1}^{T} w_t I_{[r<r_t]} \\ w_T &= (1-\lambda)/(1-\lambda^\tau) \\ w_t &= \lambda w_{t+1} \;\; t < T \end{aligned}$$

where τ is the window length.

```
var_forecast = weighted_hs(rets, lam, window, p)
```

Inputs

- `rets` (Series) t-element pandas Series with a DatetimeIndex
- `lam` (float) smoothing parameter $\in (0,1)$. Note: This parameter changed name from λ to lam due to restriction in Python
- `window` (int) the size of the window to use when computing the weighted cdf, e.g., 252 days. The estimator should be implemented using the most recent `window` observations
- `p` - (float) The level of the VaR to compute (e.g., 5%)

Outputs

- `var_forecast` - t-element pandas Series of 1-step ahead VaR forecasts. The forecast produced using data from observations $t-\tau+1,\ldots,t$ must appear in location t where τ is the `window`. The first `window-1` values must be `np.nan`.

Subsampled Realized Variance

Implement a function that estimates subsampled realized variance defined as

$$RV_{SS}^{(k)} = \frac{n/k}{n-k+1} \sum_{j=1}^{n-k+1} \left(\sum_{i=1}^{k} r_{j+i-1} \right)^2$$

where k is the desired number of highest-frequency returns (e.g., using 5 1-minute returns) in a single lower-frequency return (e.g., a 5-minute return) and n is the total number of high-frequency returns available. This estimator makes use of all consecutive blocks of return to estimate RV rather than just using disjoint (non-overlapping) blocks. The leading ratio corrects for the fact that we have too many returns in the day when we overlap. Note: The formula had a small error in the denominator and the upper limit of the first sum.

```
rv = subsampled_rv(rets, k)
```

Outputs

- `rv` (float) A scalar value containing the sub-sampled realized variance

Inputs

- `rets` - n-element pandas Series of observed high-frequency returns data covering a single day
- `k` (int) A scalar integer containing the block size

AR(1) Simulation

Implement an AR(1) simulator.

```
y = ar1_simulate(rho, errors, y0)
```

Outputs

- `y` - (1-dimensional ndarray with t elements) The simulated time series

Inputs

- `rho` - (float) Scalar autoregressive coefficient
- `errors` - (1-dimensional ndarray with t elements) t-element array of errors to use in the model
- `y0` - (float) Initial value

last page